****

**实 验 报 告**

**实验课程： 操作系统课程实践**

**学生姓名： 马星**

**学 号： 5418122020**

**专业班级： 计算机科学与技术(卓越)221班**

**2024年11月1日**

** 南昌大学实验报告**

**---（7）简单文件系统的模拟**

学生姓名： 马星 学 号： 541812020 专业班级：计算机科学与技术(卓越)221班

实验类型：□验证 □ 综合 ■设计 □ 创新 实验日期： 2024/12/20 实验成绩：

# 一、实验目的

(1)理解文件存储空间的管理、文件的物理结构和目录结构以及文件基本操作的实现。

(2)加深对文件系统内部功能和实现过程的了解。

# 二、实验内容

编写程序模拟实现一个简单的二级文件管理系统，具体可以包括以下几点：

(1)实现常见文件目录和文件操作，如：列文件、创建文件、删除文件、打开文件、关闭文件等。

(2)列文件目录时可以显示出文件名、文件长度、文件物理地址等。

# 三、实验要求

1、需写出设计说明；

2、设计实现代码及说明

3、运行结果；

# 四、主要实验步骤

## 1、 设计说明

编写程序模拟实现用户与终端的交互系统, 系统将提供常见的文件相关命令如ls（列出目录）、cd（改变当前目录）、mkdir（创建新目录）、touch（创建新文件）等

此外，用户还能够在系统中进行文件的读写操作。

在文件管理方面，系统支持多级目录的管理，并能够处理文件和目录的创建、删除、修改以及文件的读取和写入。

所有文件操作都会同步更新磁盘盘块的分配。为了高效地管理磁盘空间，使用空闲盘块链来分配和回收磁盘空间。

## 2、实现代码

### (1) 文件类

类内部记录文件(夹)的名称、路径、父目录指针

对于文件, 需要保留**输入输出流**, 提供写文件和读文件的接口;

以及记录文件的大小、给文件分配的磁盘**盘块**号

对于文件夹, 需要记录它的子文件(夹)

将文件和文件夹定义为同一个类, 用is\_folder标识类型, 这样做是因为文件夹需要引用文件(子文件), 而文件也需要引用文件夹(父目录), 防止**循环依赖**

class Folder {  
public:  
 bool is\_folder; // true:文件夹 false:文件  
 string name; // 文件(夹)名称  
 Folder \*parent; // 上级目录  
 string path; // 路径  
 // 文件属性  
 ofstream ofs; // 输出流, 用于写入文件  
 ifstream ifs; // 输入流, 用于读入文件  
 int size = 0; // 文件大小(实际大小)  
 vector<int> block;// 文件占用盘块号(盘块数\*盘块大小)  
 // 文件夹属性  
 vector<Folder \*> ChildFolder; // 子文件(夹)

### (2) 磁盘空间分配

使用**空闲盘块链**的方式分配磁盘空间, 当请求分配空间时, 从链表的首部取出相应数量的盘块分配; 当释放空间时, 将释放的盘块挂在链表尾部

初始化时用户输入(或在程序中定义)盘块数和盘块大小

为了演示程序磁盘空间不足的报错效果, 将数据调小了, 定义为8块大小为4的盘块

这里为了代码编写方便, 用队列当做空闲盘块链

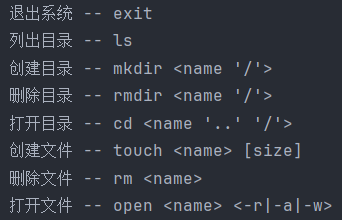
class SpaceSys {  
public:  
 static int blockCnt;// 盘块数  
 static int blockSize;// 盘块大小  
 static queue<int> blockQ;// 空闲盘块链  
 /\*\*  
 \* 初始化磁盘  
 \* @param bCnt 盘块数  
 \* @param bSize 盘块大小  
 \*/  
 static void init(int bCnt=8, int bSize=4) {  
 blockCnt = bCnt;  
 blockSize = bSize;  
 Util::info("磁盘盘块数" + to\_string(bCnt) + ", 盘块大小" + to\_string(bSize));  
 for (int i = 0; i < blockCnt; i++) {  
 blockQ.push(i);  
 }  
 }  
  
 /\*\*  
 \* 分配磁盘  
 \* @param size 文件大小  
 \* @return 分配的盘块号  
 \*/  
 static vector<int> allocation(int size) {  
 int cnt = max(1, (size + blockSize - 1) / blockSize);  
 if (blockQ.size() < cnt) {  
 Util::error("磁盘空间不足");  
 return {};  
 }  
 Util::info("分配" + to\_string(cnt) + "个盘块, 当前剩余空闲盘块" + to\_string(blockQ.size() - cnt));  
 vector<int> ans;  
 for (int i = 0; i < cnt; i++) {  
 ans.push\_back(blockQ.front());  
 blockQ.pop();  
 }  
 return ans;  
 }  
  
 /\*\*  
 \* 文件修改, 检查其空间是否足够, 空间不足则增加盘块  
 \* @param block 已分配给文件的盘块  
 \* @param size 文件当前大小  
 \* @return 空间不足时返回false  
 \*/  
 static bool reAlloc(vector<int> &block, int size) {  
 auto need = size - block.size() \* blockSize;  
 if (need <= 0) return true;// 已满足  
  
 int cnt = (need + blockSize - 1) / blockSize;  
 if (blockQ.size() < cnt) {  
 Util::error("磁盘空间不足");  
 return false;  
 }  
 Util::info("再分配" + to\_string(cnt) + "个盘块, 当前剩余空闲盘块" + to\_string(blockQ.size() - cnt));  
 for (int i = 0; i < cnt; i++) {  
 block.push\_back(blockQ.front());  
 blockQ.pop();  
 }  
 return true;  
 }  
  
 /\*\*  
 \* 释放磁盘空间  
 \* @param block 盘块号  
 \*/  
 static void releases(const vector<int> &block) {  
 Util::info("释放" + to\_string(block.size())  
 + "个盘块, 当前剩余空闲盘块" + to\_string(blockQ.size() + block.size()));  
 for (int i: block) blockQ.push(i);  
 }  
};

### (3) 终端-文件系统

终端-文件系统在程序启动时创建/加载根目录, 并用指针记录当前的目录

class FileSys {  
public:  
 Folder \*nowpath, \*root;  
  
 void init(const string &Root = "root") {  
 root = Folder::newFolder(Root, nullptr);  
 nowpath = root;  
 Util::info(root->getProjectPath());// 打印在项目中的路径(E:\.....\cmake-build-debug\root)  
 }

然后提供ls、mkdir、rmdir、cd、touch、rm、open这些命令



根据用户输入的指令进行识别, 做出相应的操作

// 终端-文件系统

FileSys fileSys;  
  
/\*\*  
\* 识别命令  
\* @param op 命令  
\* @param args 尾随参数  
\*/  
void Cmd(const string &op, const vector<string> &args) {  
 if (op == "exit") {  
 exit(0);  
 } else if (op == "ls") {  
 fileSys.ls();  
 } else if (op == "mkdir") {  
 fileSys.mkdir(args);  
 } else if (op == "rmdir") {  
 fileSys.rmdir(args);  
 } else if (op == "cd") {  
 fileSys.cd(args);  
 } else if (op == "touch") {  
 fileSys.touch(args);  
 } else if (op == "rm") {  
 fileSys.rm(args);  
 } else if (op == "open") {  
 fileSys.open(args);  
 } else {  
 Util::error("非法指令'" + op + "'");  
 }  
}

int main() {  
 SpaceSys::init(8, 4);// 8个盘块, 每个盘块大小为4 (为了演示效果,数据调小了)  
 fileSys.init();  
 cout << "退出系统 -- exit\n"  
 "列出目录 -- ls\n"  
 "创建目录 -- mkdir <name '/'>\n"  
 "删除目录 -- rmdir <name '/'>\n"  
 "打开目录 -- cd <name '..' '/'>\n"  
 "创建文件 -- touch <name> [size]\n"  
 "删除文件 -- rm <name>\n"  
 "打开文件 -- open <name> <-r|-a|-w>\n";  
 string s;  
 while (true) {  
 cout << fileSys.nowpath->getpath() << "> ";  
 getline(cin, s);// 获取用户输入的指令  
 const vector<string> &input = Util::split(s, " ");// 按空格拆分  
 Cmd(input[0], {input.begin() + 1, input.end()});// 解析指令  
 }  
}

### (4) 指令的具体实现

仅举例cd、touch、rmdir的具体实现, 不过多赘述

#### cd指令

①首先校验参数数量是否正确, cd指令仅支持1个参数, 也就是要进入的路径名称

②按照”/”分隔用户输入的路径

③维护当前路径指针, 按序遍历用户输入的路径, 如果是”..”则回退上级目录, 其他情况查找子目录, 如果找不到子目录则报错

④当找到路径后更新当前路径指针

/\*\*  
 \* 进入目录, 支持多级目录, 以"/"分隔, ".."可退至上级目录  
 \* @param args size=1 \<name>  
 \*/  
void cd(const vector<string> &args) {  
 if (args.size() != 1) {  
 Util::error("参数数量'" + to\_string(args.size()) + "'不正确");  
 return;  
 }  
 const vector<string> &dir = Util::split(args[0], "/");  
 Folder \*pNowPath = nowpath;  
 for (auto &d: dir) {  
 if (d == "..") {// 回退上级目录  
 pNowPath = pNowPath->parent;  
 } else {// 进入子目录  
 int idx = findFolder(pNowPath->ChildFolder, d);  
 if (idx == -1) {  
 Util::error("文件夹'" + d + "'不存在");  
 return;  
 }  
 pNowPath = pNowPath->ChildFolder[idx];  
 }  
 }  
 nowpath = pNowPath;  
}

#### touch指令

①首先校验参数数量是否正确

②校验文件是否存在, 若存在则询问用户是否要进行覆盖

③创建该文件, 如果用户输入了文件的预分配大小, 则进行预分配

/\*\*  
 \* 创建文件, 单个文件, 可以不写文件后缀  
 \* @param args size=1~2 <name> [size]  
 \*/  
 void touch(const vector<string> &args) {  
 if (args.empty() || args.size() > 2) {  
 Util::error("参数数量" + to\_string(args.size()) + "不正确");  
 return;  
 }  
 const string &name = args[0];  
 auto &ChildFile = nowpath->ChildFolder;  
 int idx = findFile(ChildFile, name);  
 if (idx != -1) {  
 Util::error("文件'" + name + "'已存在");  
 cout << "覆盖该文件?(Y/N):";  
 string op;  
 cin >> op;  
 if (op == "N" || op == "n") {  
 return;  
 } else if (op == "Y" || op == "y") {  
 rm(**{**name**}**);  
 } else {  
 Util::error("错误指令'" + op + "'");  
 return;  
 }  
 }  
 int size = args.size() == 2 ? stoi(args[1]) : 0;  
 Folder \*newfile = Folder::newFile(name, nowpath, size);  
 ChildFile.push\_back(newfile);  
 Util::ok();  
 }

#### rmdir指令

①首先校验参数数量是否正确, 以及要移除的目录是否存在

②del(ChildFolder[idx]), 递归删除它的子文件和文件夹

③删除这个目录以及留存在内存的指针

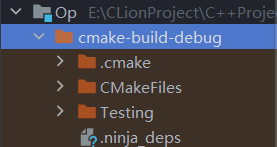
/\*\*  
 \* 递归删除a的所有子文件  
 \* @param a 文件夹  
 \*/  
void del(Folder \*a) {  
 if (!a->isFolder()) return;  
 for (auto &f: a->ChildFolder) {  
 del(f);  
 f->destroy();  
 }  
 a->ChildFolder.clear();  
}

/\*\*  
 \* 移除目录, 单级目录  
 \* @param args size=1 \<name>  
 \*/  
void rmdir(const vector<string> &args) {  
 if (args.size() != 1) {  
 Util::error("参数数量" + to\_string(args.size()) + "不正确");  
 return;  
 }  
 const string &name = args[0];  
 auto &ChildFolder = nowpath->ChildFolder;  
 int idx = findFolder(ChildFolder, name);  
 if (idx == -1) {  
 Util::error("文件夹'" + name + "'不存在");  
 return;  
 }  
 del(ChildFolder[idx]);// 删除目录的子文件夹  
 ChildFolder[idx]->destroy();// 删除目录  
 ChildFolder.erase(ChildFolder.begin() + idx);// 在内存中删除  
 Util::ok();  
}

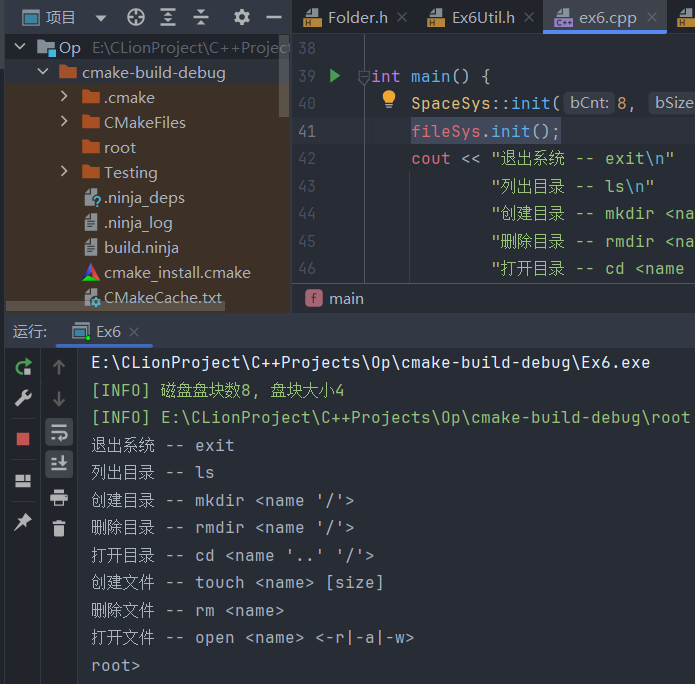
## 3、运行效果演示

### (1) 程序启动

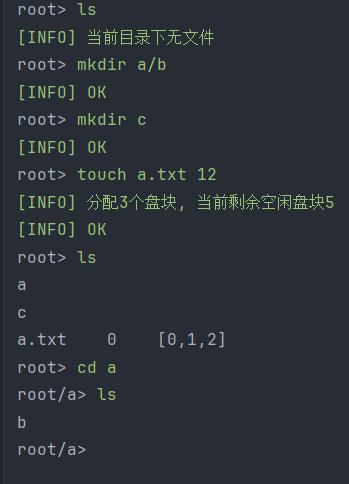
启动前, 根目录未创建

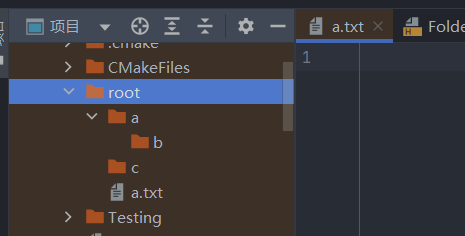


启动时创建根目录



### (2) 文件创建指令





①ls: 首先列出当前目录下的文件, 由于是首次创建的根目录, 所以目录下无文件

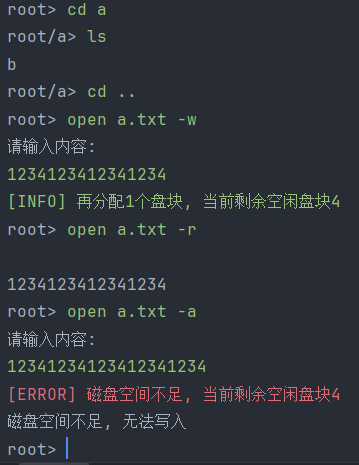
②mkdir a/b; mkdir c: 然后创建了多级目录a/b和单级目录c

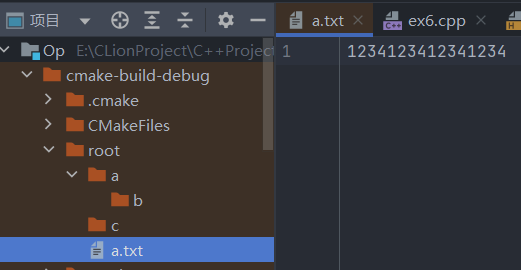
③touch a.txt 12: 创建一个预分配大小为12的文件a.txt

④ls:再次列出当前目录下的文件, 成功展示出了两个文件夹和一个文件, 其中a.txt的大小为0, 分配了3个盘块, 分配大小为3\*4=12

⑤ cd a; ls: 进入目录a, 列出a目录下的文件, 成功展示出了文件夹b

### (3) 文件读写指令





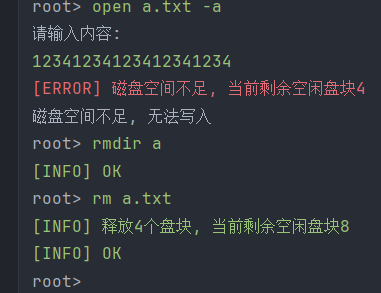
①cd .. 回退到根目录

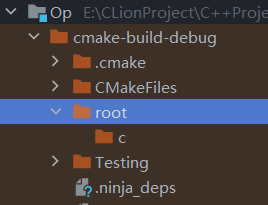
②open a.txt -w: 以覆盖写模式打开a.txt, 然后写入16大小的文本, 由于之前分配了3个盘块, 所以还需要1个盘块

③open a.txt -r: 以读模式打开a.txt, 成功展示出了a.txt的文本

④open atxt -a: 以追加写模式打开a.txr, 写入20大小的文本, 需要5个盘块, 但此时仅剩余4个盘块, 无法分配

### (4) 文件删除指令

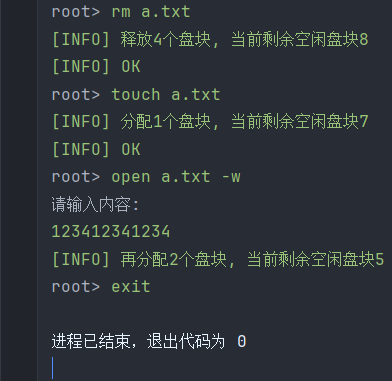




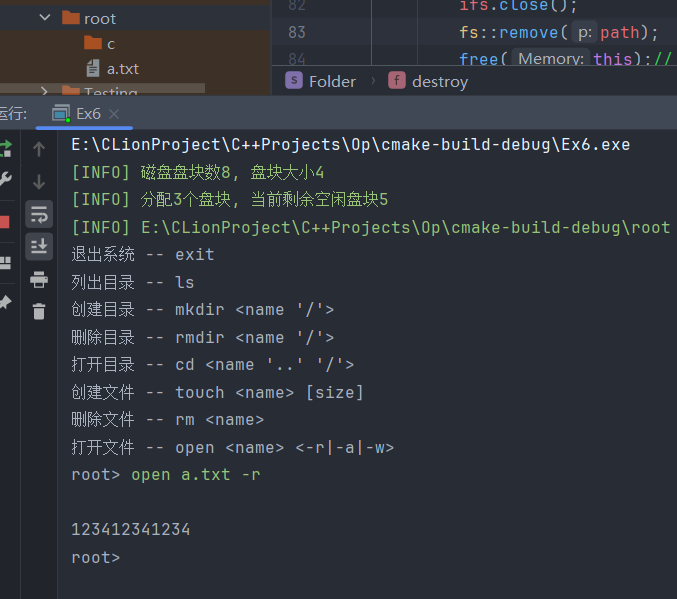
①rmdir a: 删除文件夹a和a目录下的所有子文件或文件夹

②rm a.txt: 删除文件a.txt, 并释放内存

### (5) 退出、重新启动程序



将a.txt重新创建, 写入12大小的文本, 然后退出程序



再次运行程序, 文件a.txt没有丢失, 成功被加载

由于磁盘是虚拟的, 每次都会重新初始化, 再进行盘块的分配, 没有想到一个好的模拟方案,所以磁盘这部分可能实现的不是很好

# 五、实验数据及处理结果

## (1)文件系统初始化

成功创建了根目录，并初始化磁盘空间为8个盘块，每个盘块大小为4

## (2)文件创建操作

使用mkdir命令成功创建了多级目录a/b和单级目录c

通过ls命令验证了目录结构的正确性

## (3)文件读写操作

使用touch命令创建了一个名为a.txt的文件，并预分配了12字节的空间（3个盘块）

通过open命令以覆盖写模式向a.txt写入16字节的文本，并正确分配了额外的一个盘块

通过open命令以读模式从a.txt读取并显示文本

尝试以追加写模式打开文件时由于磁盘空间不足而失败

## (4)文件删除操作

使用rmdir命令成功删除了目录a及其所有子目录和文件

使用rm命令删除了文件a.txt并释放了相应的磁盘空间

## (5)程序退出与重新启动

在程序退出前重新创建了a.txt文件，写入12字节的文本

重新启动程序后，a.txt文件被成功加载，显示了持久化存储的效果

# 六、实验体会或对改进实验的建议

## (1)磁盘空间管理

当前的磁盘空间管理策略是简单的链表形式，可以考虑实现更高效的空间分配算法，如**位图**或**索引节点映射**，以提高空间利用率和分配效率。

## (2)文件持久化存储

可以考虑引入文件系统的挂载和卸载机制，将文件系统的**状态保存**到磁盘上，以便在下次启动时恢复。

## (3)错误处理与用户反馈

当前的错误处理较为简单，可以增加更多的异常情况处理，并提供更详细的用户反馈信息，帮助用户更好地理解文件系统的状态和操作结果。

## (4)功能扩展

目前的实现只支持基本的文件和目录操作，可以进一步扩展支持**链接**、**权限管理、多端用户并发操作**等高级功能，使模拟的文件系统更加完善。

## (5)性能优化

随着文件系统规模的增大，当前的实现可能会遇到性能瓶颈。可以考虑引入**缓存机制**、**优化数据结构**等方法来提高文件系统的性能。

# 七、参考资料

《计算机操作系统实验指导》